GPU-accelerated indirect boundary element method for voxel model analyses with fast multipole method

AUTHOR(S):

Hamada, Shoji

CITATION:

Hamada, Shoji. GPU-accelerated indirect boundary element method for voxel model analyses with fast multipole method. Computer Physics Communications 2011, 182(5): 1162-1168

ISSUE DATE:

2011-05

URL:

http://hdl.handle.net/2433/157328

GPU-accelerated indirect boundary element method for voxel model analyses with fast multipole method

Shoji Hamada

Department of Electrical Engineering, Kyoto University, Kyoto-daigaku-katsura, Kyoto 615-8510, Japan

Abstract

An indirect boundary element method (BEM) that uses the fast multipole method (FMM) was accelerated using graphics processing units (GPUs) to reduce the time required to calculate a three-dimensional electrostatic field. The BEM is designed to handle cubic voxel models and is specialized to consider square voxel walls as boundary surface elements. The FMM handles the interactions among the surface charge elements and directly outputs surface integrals of the fields over each individual element. The CPU code was originally developed for field analysis in human voxel models derived from anatomical images. FMM processes are programmed using the NVIDIA Compute Unified Device Architecture (CUDA) with double-precision floating-point arithmetic on the basis of a shared pseudocode template. The electric field induced by DC-current application between two electrodes is calculated for two models with 499,629 (model 1) and 1,458,813 (model 2) surface elements. The calculation times were measured with a four-GPU configuration (two NVIDIA GTX295 cards) with four CPU cores (an Intel Core i7-975 processor). The times required by a linear system solver are 31 s and 186 s for models 1 and 2, respectively. The speed-up ratios of the FMM range from 5.9 to 8.2 for model 1 and from 5.0 to 5.6 for model 2. The calculation speed for element-interaction in this BEM analysis was comparable to that of particle-interaction using FMM on a GPU.

Keywords: Boundary element method, Fast multipole method, Graphics processing unit, Voxel model, Laplace equation

1. Introduction

Graphics processing units (GPUs) are increasingly being applied to scientific computations because they offer a high performance–cost ratio and a low barrier to entry. NVIDIA is one of the most influential promoters of general-purpose computing on GPUs with its computing architecture called Compute Unified Device Architecture (CUDA)[1]. Useful results have been obtained using GPUs for N-body problems [2, 3] both with and without fast algorithms such as the fast multipole method (FMM) [4, 5] and the tree method. The speed-up ratio when using GPUs instead of CPUs with single-precision floating-point arithmetic occasionally exceeds one hundred [3]. GPUs have also been demonstrated to significantly accelerate the computation of boundary element methods (BEMs) [6, 7], although they have not yet been applied to fast algorithms.

In a previous study, the author developed an indirect BEM accelerated by the FMM; this approach was designed to handle cubic voxel models and was specialized to consider square voxel walls as boundary surface elements [8, 9]. Note that the indirect BEM is also referred to as the equivalent source method [10], surface charge simulation method [11], and method of moments (MoM) [12]. In this approach, the Laplace kernel FMM handled the interactions among surface charge elements instead of point charges, and it directly provided the surface integrals of the fields over each individual element. Using the BEM, three-dimensional electric fields were analyzed in human voxel models derived from anatomical images. The quality of calculated fields was similar to that produced by the scalar-potential finite-difference method, impedance method, and quasi-static finite-difference time-domain method [13], indicating practical applicability and usefulness. This method has the following features: (1) the calculated fields macroscopically satisfy Gauss's law, and (2) the FMM embedded in an iterative solver only performs multiply-and-accumulate operations.

In the current study, the author programmed the indirect FMM-BEM on GPUs using CUDA with double-precision floating-point arithmetic [14]. The FMM utilizes rotation-coaxial translation-rotation (RCR) decomposition [5, 15] of the multipole to local translation (M2L) operator. The processes of the FMM are coded on the basis of a pseudocode template by adopting a strategy of one CUDA block per FMM box. This code was used to calculate the DC conductive current and electric field in two human head models in which a current was applied through two contact electrodes. This type of field analysis is required in studies of electric current dosimetry and

Email address: shamada@kuee.kyoto-u.ac.jp (Shoji Hamada)

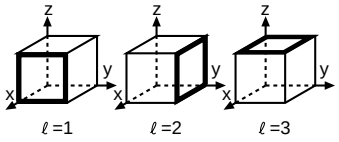

Figure 1: Voxel surfaces indicated by index ℓ.

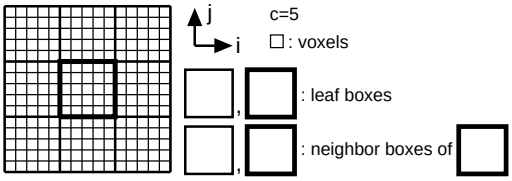

Figure 2: Voxels, leaf boxes, and neighboring boxes.

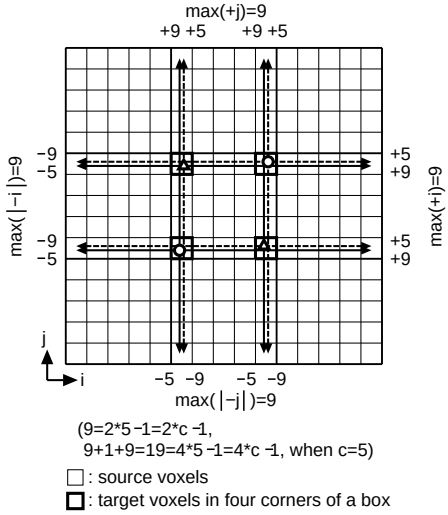

Figure 3: Arrangement of source and target voxels in direct field calculation.

in electroencephalogram signal analyses. The calculation times with four GPUs and four CPU cores were measured on a personal computer (PC). To the best of the author's knowledge, this is the first GPU version of the indirect FMM-BEM for voxel model analysis.

2. Indirect BEM for voxel model analysis

2.1. Overview [8]

Consider a rectangular region composed of $N_x \times N_y \times N_z$ cubic voxels, where the voxel side length is L. A set of integer indices defined as (i, j, k)=(0:N_x−1, 0:N_y−1, 0:N_z−1) indicates voxel positions in (x, y, z) coordinates, where "a:b" means that the index ranges from a to b. It is assumed that each voxel at (i, j, k) has homogeneous conductivity $\sigma(i, j, k)$, which defines a conductive volume model. Another integer index ℓ=1:3 is defined to indicate a square surface on a voxel (see Fig. 1), then a set of indices (i, j, k, ℓ) identifies a voxel surface in the region. When a voxel surface resides between two voxels having different conductivities, the surface is considered as a square boundary element. This procedure produces a unique surface model. It is assumed that each element has unknown homogeneous charge density q generating electric field $\boldsymbol{E}$ and potential ϕ. All unknown q are determined by solving a linear system composed of boundary equations. Then the field is determined at each voxel center.

On the basis of the MoM approach, boundary equations are represented with $\int \boldsymbol{E}_{\pm} \cdot \boldsymbol{n} dS \equiv F_{\pm}$ and $\int \phi dS \equiv V$, where $\int dS$ gives the surface integral on an element, $\boldsymbol{n}$ is a unit normal vector of the element (i.e., $\boldsymbol{n}$=$\boldsymbol{i}$, $\boldsymbol{j}$, or $\boldsymbol{k}$), and the subscript $\pm$ indicates that $\boldsymbol{E}$ is defined on the plus or minus side of the element in the x-, y-, and z-coordinates (see Appendix A). The q, $F_{\pm}$, and V are also uniquely identified by the indices (i, j, k, ℓ).

To efficiently calculate all $F_{\pm}$ and V, the FMM is applied. A cubic box covering the entire model region is first defined at a root level. This box is repeatedly partitioned into eight smaller cubic boxes defined at a finer level, which generates a hierarchical oct-tree box structure. The finest-level boxes are called the leaf boxes, and these are naturally defined as clusters of c^3 voxels in cubic voxel models. In this paper, c ranges from 5 to 7. Fig. 2 shows a two-dimensional diagram of leaf boxes for c=5. The F and V are calculated using two complementary approaches. One approach is the direct field calculation that considers nearby elements in neighboring boxes, which are denoted by F^{direct} and V^{direct}. The set of neighboring boxes is defined for each box, and the box itself is included in the set. Fig. 2 also shows a two-dimensional diagram of neighboring boxes. The other approach is a non-straightforward field calculation with multipole and local expansion coefficients representing the contributions of far elements, which are denoted by F^{far} and V^{far}.

2.2. Direct field calculation for voxel model analysis

Symbols "s" and "t" are defined to indicate the role of a surface as source or target, respectively. For example, $(is, js, ks, \ell s)$ identifies a source element. Further, the relative arrangement of a target element observed from a source element is defined as $(it{-}is,\ jt{-}js,\ kt{-}ks,\ \ell t,\ \ell s)$. In the calculation of F^{direct} and V^{direct}, the possible relative arrangement is restricted to $(-2c{+}1{:}2c{-}1,\ -2c{+}1{:}2c{-}1,\ -2c{+}1{:}2c{-}1,\ 1{:}3,\ 1{:}3)$. Fig. 3 shows a two-dimensional diagram of possible arrangements for two voxels, one containing a source and the other a target, indicating that there are $-2c{+}1{:}2c{-}1$ relative arrangements per dimension. Therefore, the number of independent interaction coefficients is $3{\times}3{\times}(4c{-}1)^3$, which can be preliminarily calculated as unit source responses f^{direct} and v^{direct}. Both f_{+}^{direct} and f_{-}^{direct} have the same value except when the target and source are identical. Conversely, when they are

identical, the following relationship is satisfied:

$$f_{\pm}^{\text{direct}}(0,0,0,\ell t,\ell s) = \pm\frac{S}{2\varepsilon_0} \quad (\ell t=\ell s=1:3), \quad (1)$$

where S=L^2, and ε_0 is the permittivity of free space. The consolidated F^{direct} and V^{direct} are calculated using the following multiply-and-accumulate operations:

$$F^{\text{direct}}(it,jt,kt,\ell t) = \sum_s q(is,js,ks,\ell s) \times f^{\text{direct}}(it-is,jt-js,kt-ks,\ell t,\ell s) \quad (2)$$

$$V^{\text{direct}}(it,jt,kt,\ell t) = \sum_s q(is,js,ks,\ell s) \times v^{\text{direct}}(it-is,jt-js,kt-ks,\ell t,\ell s), \quad (3)$$

where $\sum_s$ denotes the summation of the contributions from all related sources.

2.3. Far field calculation for voxel model analysis

Multipole expansion coefficients M_n^m defined on a leaf box consist of independent real and pure imaginary numbers M_{ic}, where $ic = 0$ to $(p+1)^2-1$, when the expansion is truncated to p. The formula for calculating M_n^m produced by a point charge in a leaf box is well known [5]. Given a source surface element with unit charge density in a leaf box, M_{ic} can be calculated by numerical integration with this formula, and these unit responses are stored as m_{ic}. The number of possible source element positions is $3\times c^3$. Thus, preliminary calculations of m_{ic} generated by these sources provide a full set of unit responses. The consolidated M_{ic} on a leaf box is calculated by the following multiply-and-accumulate operation (Q2M):

$$M_{ic} = \sum_s q(is',js',ks',\ell s)\times m_{ic}(is',js',ks',\ell s), \quad (4)$$

where $(is', js', ks', \ell s)$=(0:c−1, 0:c−1, 0:c−1, 1:3) indicates the local element positions in a leaf box. Local expansion coefficients L_n^m and L_{ic} are defined on a leaf box. The formulae for calculating $\boldsymbol{E}$ and ϕ at a point in the leaf box produced by L_n^m are well known [5]. Given a unit L_{ic} as a source, F^{far} and V^{far} on a target element can be calculated by numerical integration with these formulae, and these unit responses are stored as f_{ic}^{far} and v_{ic}^{far}, respectively. Preliminary calculations of f_{ic}^{far} and v_{ic}^{far} at $3\times c^3$ target positions provide full sets of unit responses. The consolidated F^{far} and V^{far} are calculated by the following multiply-and-accumulate operations (L2F and L2V):

$$F^{\text{far}}(it',jt',kt',\ell t) = \sum_{ic=0}^{(p+1)^2-1} L_{ic}\times f_{ic}^{\text{far}}(it',jt',kt',\ell t), \quad (5)$$

$$V^{\text{far}}(it',jt',kt',\ell t) = \sum_{ic=0}^{(p+1)^2-1} L_{ic}\times v_{ic}^{\text{far}}(it',jt',kt',\ell t). \quad (6)$$

The other FMM processes in the far field calculation are identical to those in the FMM adopting RCR decomposition of translation operators [5], which are the following translations: multipole-to-multipole (M2M), multipole-to-local (M2L), and local-to-local (L2L).

Table 1: Pseudocode template for GPU kernels of FMM.

```
01: For Loop 1: handle P sub-processes {
02:     Store fixed constants in shared memory, if possible
03:     Process FMM boxes via 30 CUDA blocks {
04:       Store fixed source data in shared memory, if possible
05:       Process targets via T CUDA threads {
06:         For Loop 2: handle sources related to each target{
07:           Multiply-and-accumulate related
              source contributions
08:         }// end For
09:       }// Multiple threads jointly handle a target, if possible.
10:       Do reduction when multiple threads
          jointly handle a target above
11:     }// CUDA blocks always handle the same boxes.
12: }// end For
```

3. CUDA kernel template

The FMM processes were programmed for GPUs using CUDA with double-precision floating-point arithmetic. The CUDA kernels, which are functions that a GPU executes, are coded on the basis of a pseudocode template. The use of a shared template for all FMM procedures makes the coding easier.

The specifications of the PC used, which constitute the fixed hardware parameters, are summarized as follows: The operating system, CPU, and GPUs were 64-bit Microsoft Windows Vista, Intel Core i7-975 (four CPU cores, 3.33 GHz), and two NVIDIA GTX295 cards (four GPUs), respectively. Note that a GTX295 card has two GPUs; this card is capable of double-precision floating-point arithmetic and has 940 MB of GDDR3 global memory per GPU, 30 multiprocessors per GPU, and 16 kB shared memory per multiprocessor. To store f^{direct}, v^{direct}, m_{ic}, f_{ic}^{far}, and v_{ic}^{far} in double-precision arrays, a memory capacity greater than 16 kB is required. The memory size of f^{direct} is 493 kB (c=5), 876 kB (c=6), or 1417 kB (c=7), and that of v^{direct} is the same. In this paper, p is fixed to 10 [3]; therefore, ic ranges from 0 to 120, and the memory size of m_{ic} is 363 kB (c=5), 627 kB (c=6), or 996 kB (c=7). Those of f_{ic}^{far} and v_{ic}^{far} are the same. On the other hand, when p is 10, the memory size requirement of the rotation operator is 14,168 bytes, that of the coaxial-translation operator in M2M and L2L is 2288 bytes, and that in M2L is 4048 bytes.

Table 1 shows the pseudocode template for the CUDA kernels. In the template, the CUDA block can be regarded as a multiprocessor on a GPU in the context of this study; thus, the number of CUDA blocks is set to 30 (line 03 in Table 1). The template adopts a strategy of one CUDA block per box (line 03). Each CUDA block has its shared memory, and T CUDA threads per CUDA block execute multithreading multiply-and-accumulate operations (lines 05-09) by sharing data in the shared memory. The CUDA blocks and CUDA threads are used in one-

Table 2: Parameters of GPU kernels of FMM.

Process	P	T	Sequential sub-parts	Boxes (blocks)	Sources	Targets (threads)
Direct field (c=5, 6)	243	256	F^{direct} evaluation V^{direct} evaluation	Leaves	q	F^{direct} V^{direct}
Direct field (c=7)	972	256	F^{direct} evaluation V^{direct} evaluation	Leaves	q	F^{direct} V^{direct}
Q2M	1	256		Leaves	q	M_{ic}
M2M	8	256	forward-rotation coaxial-translation backward-rotation	Parents	M_{ic}	M_{ic}
M2L	316	128	forward-rotation coaxial-translation backward-rotation	All	M_{ic}	L_{ic}
L2L	8	256	forward-rotation coaxial-translation backward-rotation	Parents	L_{ic}	L_{ic}
L2F L2V	1	256	F^{far} evaluation V^{far} evaluation	Leaves	L_{ic}	F^{far} V^{far}

dimensional forms, as done by Gumerov and Duraiswami [2] and Yokota et al. [3]. Because efficient use of shared memory is one of the keys to high-performance GPU computation, some FMM processes are divided into P sub-processes (line 01), thus dividing unit responses into subsets smaller than the shared memory capacity (line 02). In particular, the direct field calculation process and M2L process, which are the most time-consuming, are divided into several hundreds of sub-processes. Table 2 summarizes P and the other parameters in the template, as well as the sequential sub-parts involved in each process.

4. CUDA kernel

4.1. Direct field calculation kernel

Direct field calculation based on Eqs. (2) and (3) is performed as follows: Surface elements are accessed via leaf boxes. When a leaf box is handled as a target box, the contained elements are regarded as target elements, and F^{direct} and V^{direct} on these targets are to be calculated. Elements in neighboring boxes to the target box are regarded as source elements. In the GPU environment, this process is divided into P sub-processes to divide f^{direct} and v^{direct} into subsets that are loadable into shared memory.

For c=5 or 6, a leaf box is treated as a set of three sub-leaf boxes. Elements are sorted into them according to the value of ℓ. Target elements (ℓt=1, 2, and 3) have to gather the contributions of the source elements (ℓs=1, 2, and 3) contained in 3×3×3 neighbor boxes. By dividing this gathering process into 3×3×(3×3×3)=243 sub-processes, the number of entries in the divided f^{direct} or v^{direct} is reduced to $(2c-1)^3$. Fig. 4 shows a two-dimensional diagram of possible arrangements of source and target voxels, which indicates that there are $2c-1$ relative arrangements per dimension. The corresponding memory size is 5832 bytes (c=5) or 10,648 bytes (c=6).

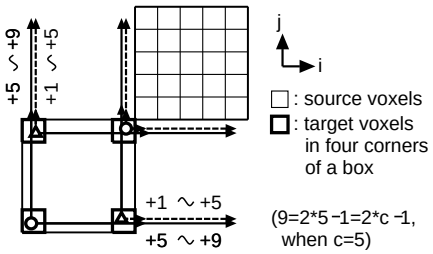

Figure 4: Arrangement of source and target voxels in a case of divided direct field calculation.

For c=7, much more size reduction is required. A local index k''=0:6 is defined in a leaf box, which is an alias of k. Lower and upper sub-boxes are defined to cover k''=0:3 and 3:6, respectively, with an overlap at k''=3. These two sub-boxes and ℓ define six new sub-leaf boxes. Elements are sorted into them according to ℓ and k''. Auxiliary rules for sorting elements at k''=3 are designed by considering calculation efficiency. By dividing the gathering process into 6×6×(3×3×3)=972 sub-processes, the number of entries and memory size are reduced to $(2c-1)^2(2c'-1)$ and 9464 bytes, respectively, where c'=4.

Steps (I-1)–(I-7) below are performed two times, once each for the evaluation of F^{direct} and V^{direct}.

(I-1) For Loop 1 in Table 1 handles P sub-processes.

(I-2) When a sub-process is fixed, the corresponding divided v^{direct} or f^{direct} is loaded into shared memory.

(I-3) Thirty CUDA blocks handle all sub-leaf boxes in parallel. To avoid race conditions, a CUDA block always handles the same sub-leaf boxes through all sub-processes. Elements in the handled sub-leaf box act as targets. Elements in the paired sub-leaf box specified by current sub-process act as sources. If either box of the pair has no elements or is undefined, the subsequent steps are skipped.

(I-4) The local position and charge density of the sources are loaded into shared memory.

(I-5) Both the number of targets (n_{t}) and the number of sources (n_{s}) are at most 5×5×5=125 (c=5), 6×6×6=216 (c=6), or 7×7×4=196 (c=7). The number of CUDA threads T is set to 256 ($>n_{\mathrm{t}}$). A target is jointly handled by multiple CUDA threads. The number of these threads is automatically set as 1, 2, 4, 8, or 16 on the basis of both n_{t} and n_{s}.

(I-6) Each thread gathers related source contributions into shared memory via For Loop 2 in Table 1.

(I-7) After reduction of values in shared memory, they are stored in global memory as F^{direct} or V^{direct}.

4.2. Far field calculation kernel

All processes of the far field calculation are coded using the template in Table 1. The details of the CUDA kernels for Q2M, L2F, and L2V based on Eqs. (4), (5), and (6),

respectively, are described below. Those of M2M, M2L, and L2L are omitted. In the present codes, unit responses m_{ic}, f_{ic}^{far}, and v_{ic}^{far} are not loaded into shared memory.

4.2.1. Q2M

(II-1) For Loop 1 handles a single sub-process.

(II-2) No operation.

(II-3) Thirty CUDA blocks handle all leaf boxes in parallel. M_{ic} of a leaf box act as targets. Elements in the leaf box act as sources, n_{s} is at most $3c^3$: 375 (c=5), 648 (c=6), or 1029 (c=7).

(II-4) The local position and charge density of the sources are loaded into shared memory.

(II-5) n_{t}=121 because ic=0–120. T is set to 256 ($>2n_{\mathrm{t}}$). Two threads jointly handle a target, and they share the related sources as evenly as possible.

(II-6) Each thread gathers related source contributions into shared memory via For Loop 2.

(II-7) M_{ic} are stored in global memory after the reduction of the two values gathered in shared memory.

4.2.2. L2F and L2V

Steps (III-1)–(III-7) below are performed two times, once each for the evaluation of F^{far} and V^{far}.

(III-1) For Loop 1 handles a single sub-process.

(III-2) Auxiliary integer arrays for array address calculation are loaded into shared memory.

(III-3) Thirty CUDA blocks handle all leaf boxes in parallel. L_{ic} act as sources (n_{s}=121); elements in each leaf box act as targets.

(III-4) Sources L_{ic} are loaded into shared memory.

(III-5) n_{t} is at most $3\times c^3$: 375 (c=5), 648 (c=6), or 1029 (c=7). T is set to 256, and one thread handles a target. When n_{t} is greater than 256, the CUDA threads sequentially handle multiple targets.

(III-6) Each thread gathers 121 source contributions for a target via For Loop 2.

(III-7) Calculated F^{far} or V^{far} are stored in global memory.

4.3. Multiple GPU execution

The number of GPUs, N_g, is at most four in this study. N_g CPU threads are created by Open Multi-Processing (OpenMP) to control N_g GPUs in parallel. Fig. 5 shows a block diagram of multiple GPU execution for N_g=4. Direct field calculation and M2L are performed using N_g GPUs. The other processes are performed by one GPU with a GPU-id (i.e., identification number in the range 0 to N_g-1) of 0. The allocation and content of the global memory on each GPU are set identical to those in single-GPU execution. Thus, no memory saving is considered. P sub-processes are divided into N_g subsets as evenly as possible. N_g GPUs share the subsets one by one as "for (int i=GPU-id; $i<P$; $i=i+N_g$)", where i is a unique sub-process number. This approach is expected to achieve good load balancing and reduced calculation time. Data

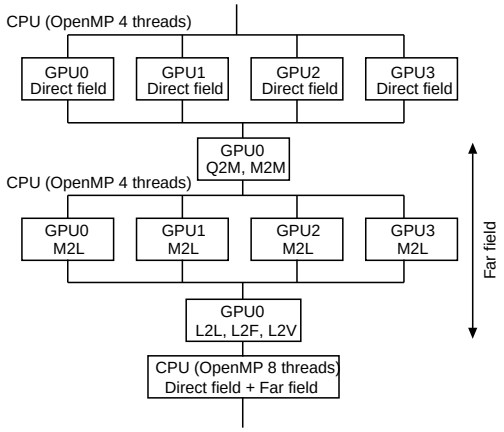

Figure 5: Block diagram of FMM on four GPUs.

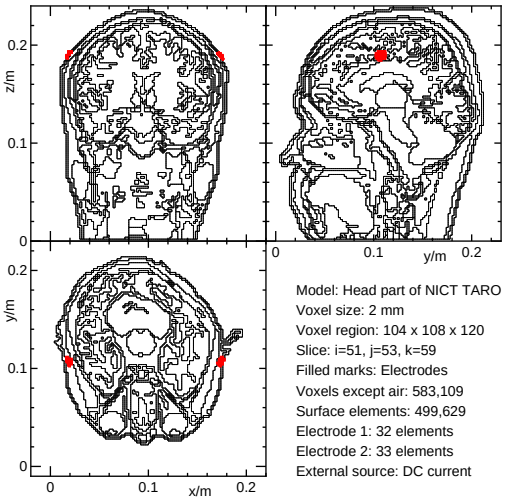

Figure 6: Voxel head model 1 and electrode arrangement.

transfer before and after M2L is performed as follows: after M_{ic} is calculated in M2M on a GPU (GPU-id=0), it is transferred from the GPU to the CPU, and then from the CPU to other GPUs. After calculation of N_g segments of L_{ic} in M2L on N_g GPUs, they are transferred from the GPUs to the CPU, consolidated on the CPU, and then transferred to the GPU (GPU-id=0).

5. Voxel models and solver

The developed code was used to calculate the DC conductive current and electric field in two anatomical human head voxel models in which electric current is applied through two contact electrodes.

Model 1, shown in Fig. 6, is part of a Japanese adult male model called TARO, developed by the National Institute of Information and Communications Technology (NICT) [16]. The length L is 2 mm, and (N_x, N_y, N_z)=(104, 108, 120). Model 1 contains 583,109 non-air voxels, 499,629

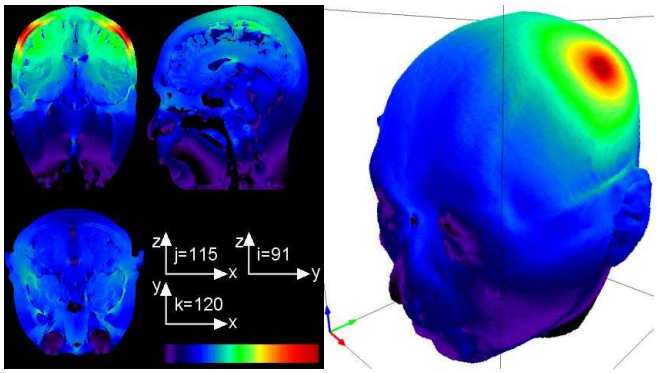

Figure 7: Distribution of $|\boldsymbol{E}|$ in model 2.

Table 3: Calculation time and number of iterations of the CPU version with model 1.

c	5	6	7
Time for direct field	0.916 s	1.396 s	2.215 s
Time for M2L	1.481 s	0.902 s	0.581 s
Time for far field	1.551 s	0.962 s	0.644 s
Half-step time	2.478 s	2.369 s	2.870 s
Iterations	53	52	48
Total time	263.2 s	246.9 s	276.3 s

surface elements, and 22 types of isotropically conductive tissues. Model 2 is an original head model derived from magnetic resonance images of a Japanese adult male. The length L is 1 mm, and $(N_x,\ N_y,\ N_z)$=(184, 232, 241). This model contains 4,625,755 non-air voxels, 1,458,813 surface elements, and 9 types of tissues. The conductivity values used for the tissues are those used by Hirata et al. [13]. Two circular plate electrodes are attached to the skin surface; such electrodes are composed of surface elements identical to the attached skin elements. The radius of the electrodes is set to 5 mm. In Fig. 6, the electrodes are drawn as filled areas projected onto the slice planes.

A biconjugate gradient method called the BiCGSafe method [17] is utilized to iteratively solve the linear system for 499,629 or 1,458,813 unknown charge densities. To regularize the linear system, the total charge is fixed at zero. Convergence is judged when the relative residual norm of the solution becomes less than 10^{-6}. Fig. 7 shows an example of the obtained $|\boldsymbol{E}|$ distribution in model 2 on three slices and on the skin surface. The color gradation ranges from $\log_{10}\left(10^{-4}\,|\boldsymbol{E}_{\max}|\right)$ to $\log_{10}\left(|\boldsymbol{E}_{\max}|\right)$.

6. Results

6.1. Analysis of field in model 1

The electric field in model 1 was analyzed by changing the leaf box size c^3 from 5^3 to 7^3. The leaf level is 5 in all cases when the root level is counted as 0.

Table 4: Calculation time, number of iterations, and speed-up of the four-GPU version with model 1.

c	5	6	7
Time for direct field	0.110 s (8.36×C)	0.106 s (13.2×C)	0.200 s (11.1×C)
Time for M2L	0.228 s (6.51×C)	0.135 s (6.69×C)	0.086 s (6.78×C)
Time for far field	0.291 s (5.32×C)	0.186 s (5.17×C)	0.133 s (4.83×C)
Half-step time	0.417 s (5.94×C)	0.309 s (7.68×C)	0.350 s (8.21×C)
Iterations	49	50	50
Total time	41.6 s	31.3 s	35.5 s

A CPU-version code was executed to provide reference values with eight OpenMP threads. Table 3 shows the calculation times and number of iterations. The time required for M2L is a part of that for the far field, and the half-step time is nearly equal to the sum of the times required for the direct field and the far field. Thus, the half-step time is nearly equal to the FMM calculation time. The fastest half-step time was obtained for c=6. The total time is nearly equal to the product of the number of iterations required for convergence and twice the half-step time, because the solver, BiCGSafe, performs two FMM calculations per iteration step. The total times range from 247 s to 276 s.

Table 4 shows the calculation times, number of iterations, and speed-up ratios observed with the four-GPU version. The speed-up ratios are noted in parentheses with "×C" to emphasize that these results are relative to those obtained with the CPU version. The speed-up ratios of the direct field, far field, and half-step time range from 8.4 to 13.2×C, from 4.8 to 5.3×C, and from 5.9 to 8.2×C, respectively. Those of M2L range from 6.5 to 6.8×C, which are better than those of the far field. This implies that the single-GPU parts in the far field process reduce the speed-up ratio by 20%–30%. The fastest half-step time, 0.309 s, was also obtained for c=6, and the best speed-up ratio of the half-step time, 8.21×C, was obtained for c=7. The total times range from 31 s to 42 s.

Table 5 lists the calculation times and number of iterations observed in the execution of the 1-, 2-, 3-, and 4-GPU versions. The speed-up ratios are noted in parentheses against those of the single-GPU version; these ratios are indicated by "×S." Increasing the number of GPUs results in an increase in the speed-up ratio. Speed-up ratios of direct field and M2L are 3.7×S and 3.6×S with four GPUs, respectively; however, those of far field and half-step time are 2.9×S and 3.1×S, respectively, indicating that the single-GPU parts in the far field process reduce the speed-up ratio by approximately 20%. Note that the value 3.1×S for the half-step time indicates that it still attains approximately 77% of the ideal speed-up ratio with four GPUs.

Table 5: Calculation time, number of iterations, and speed-up in relation to the number of GPUs with model 1 (c=6).

GPUs	1	2	3	4
Time for direct field	0.392 s	0.200 s (1.96×S)	0.139 s (2.81×S)	0.106 s (3.70×S)
Time for M2L	0.490 s	0.246 s (1.99×S)	0.167 s (2.93×S)	0.135 s (3.63×S)
Time for far field	0.538 s	0.297 s (1.81×S)	0.217 s (2.48×S)	0.186 s (2.89×S)
Half-step time	0.943 s	0.511 s (1.84×S)	0.372 s (2.54×S)	0.309 s (3.06×S)
Iterations	49	52	53	50
Total time	92.7 s	53.5 s	39.8 s	31.3 s

Table 6: Calculation time and number of iterations of the CPU version with model 2.

c	5	6	7
Time for direct field	1.637 s	2.195 s	3.811 s
Time for M2L	8.287 s	5.340 s	3.556 s
Time for far field	8.611 s	5.593 s	3.761 s
Half-step time	10.28 s	7.821 s	7.605 s
Iterations	61	60	70
Total time	1254.8 s	939.1 s	1065.3 s

6.2. Analysis of field in model 2

The electric field in model 2 was also analyzed in the same manner as that for model 1. The leaf level is 6 in all cases. Table 6 lists the calculation times and number of iterations observed in the CPU version executed with eight OpenMP threads. The fastest half-step time was obtained for c=7. The total times range from 939 s to 1255 s.

Table 7 lists the calculation times, number of iterations, and speed-up ratios observed in the execution of the four-GPU version. The speed-up ratios of the direct field, far field, and half-step time range from 3.8 to 6.6×C, from 5.3 to 5.5×C, and from 5.0 to 5.6×C, respectively. Those of M2L range from 6.3 to 6.5×C, indicating that the single-GPU parts in the far field process reduce the speed-up ratio by approximately 15%–20%. The fastest half-step time, 1.407 s, was obtained for c=6, and the best speed-up ratio of the half-step time, 5.56×C, was also obtained for c=6. The total times range from 186 s to 253 s.

Table 7: Calculation time, number of iterations, and speed-up of the four-GPU version with model 2.

c	5	6	7
Time for direct field	0.426 s (3.84×C)	0.331 s (6.64×C)	0.725 s (5.26×C)
Time for M2L	1.317 s (6.29×C)	0.824 s (6.48×C)	0.553 s (6.44×C)
Time for far field	1.591 s (5.41×C)	1.025 s (5.46×C)	0.717 s (5.25×C)
Half-step time	2.069 s (4.97×C)	1.407 s (5.56×C)	1.494 s (5.09×C)
Iterations	61	66	66
Total time	253.2 s	186.4 s	198.0 s

Table 8: Calculation time, number of iterations, and speed-up in relation to the number of GPUs with model 2 (c=7).

GPUs	1	2	3	4
Time for direct field	2.817 s	1.417 s (1.99×S)	0.957 s (2.94×S)	0.725 s (3.89×S)
Time for M2L	2.028 s	1.024 s (1.98×S)	0.690 s (2.94×S)	0.553 s (3.67×S)
Time for far field	2.179 s	1.178 s (1.85×S)	0.849 s (2.56×S)	0.717 s (3.04×S)
Half-step time	5.035 s	2.639 s (1.91×S)	1.854 s (2.72×S)	1.494 s (3.37×S)
Iterations	65	63	69	66
Total time	655.0 s	333.0 s	256.5 s	198.0 s

Table 9: Comparison of calculation times of FMM by a single GPU.

	Reference [2]		Reference [3]	Present	
GPU(GeForce)	8800GTX		8800GT	GTX295	
Core clock	575 MHz		600 MHz	576 MHz	
Multiprocessors	16		14	30/GPU	
Precision	Single		Single	Double	
Bodies	Particles		Particles	Surfaces	
Distribution	Random		Random	Model surface	
$N/10^6$	~ 1.0		~ 1.0	~ 0.5	~ 1.5
p	7	11	10	10	10
Time	∼0.9 s	∼1.4 s	∼8.0 s	0.94 s	5.04 s

Table 8 lists the calculation times, number of iterations, and speed-up ratios observed in the execution of the 1-, 2-, 3-, and 4-GPU versions. The speed-up ratios of direct field and M2L are 3.9×S and 3.7×S with four GPUs, respectively; however, those of far field and half-step time are 3.0×S and 3.4×S, indicating that the single-GPU parts in the far field process reduce the speed-up ratio by approximately 20%. The value 3.4×S in the half-step time indicates that it still achieves approximately 84% of the ideal speed-up ratio with four GPUs.

Although the multiple-GPU procedures described in section 4.3 were adopted in this study, various alternative procedures can also be adopted. Partitioning of FMM boxes into N_g subsets makes it possible to execute all FMM processes with N_g GPUs [3]. This method will be suited to reducing both memory and time, including data-transfer time. Note that this technique can be embedded into the pseudocode template in Table 1.

6.3. Comparison of FMM calculation times

The speed-up ratios obtained with the FMM on GPUs compared with CPU cores presented in previous sections range from 5.0 to 8.2×C, which might not be noteworthy when compared with those attained in preceding studies [2, 3]. To investigate these results from another perspective, the calculation times for a single FMM execution by a single GPU are summarized in Table 9, which shows values of approximately the same order. It is not the author's intention to portray an exact comparison of these times, because both the specifications of the utilized GPU and the handled problems are not the same. Nevertheless, in general, the calculation speed is almost proportional to the core clock speed and the number of multiprocessors. The double-precision floating-point operations take significantly longer time than single-precision operations. Further, the truncation number p increases the far field calculation time in proportion to $(p+1)^3$ [5] under the definition adopted in this study. In Table 9, the values of p are adjusted to conform to this definition. In addition, the FMM in this study handles the interactions among surface elements and outputs surface integrals over individual elements, which is considerably different from the approach in the other studies. From Table 9, it is concluded that this study achieved BEM analyses on the basis of calculation of the interactions among surface elements by the FMM on GPUs, and the calculation speed achieved is comparable to those in the preceding random particle-interaction studies.

7. Summary

An indirect boundary element method (BEM) with the fast multipole method (FMM) was accelerated by implementation on graphics processing units (GPUs) to reduce the calculation time of three-dimensional electrostatic fields. The BEM code is designed to handle cubic voxel models and specialized to consider voxel walls as square boundary elements. The FMM handles the interaction among the surface charge elements and directly outputs the surface integrals of the fields over individual elements. The processes of the FMM were programmed using NVIDIA CUDA with double-precision (DP) floating-point arithmetic on the basis of a pseudocode template. Two anatomical head models were utilized to calculate the fields induced by DC current application between two attached electrodes. The models had 499,629 (model 1) and 1,458,813 (model 2) surface elements. Calculations were performed on a PC with four GPUs (two NVIDIA GTX295 cards; 75G or 112G peak DP flops per GPU) and four CPU cores (one Intel Core i7-975; 55G peak DP flops). In the analysis of model 1 with four GPUs, the time required for one FMM execution, its speed-up ratio relative to four CPU cores, and the calculation time required by a solver are 0.31 s, 5.9–8.2 times, and 31 s, respectively. In the analysis of model 2 with four GPUs, these respective values are 1.4 s, 5.0–5.6 times, and 186 s, respectively. In the analysis of models 1 and 2 using a single GPU, the time required for one FMM execution for element-interaction calculation is 0.94 s and 5.0 s, respectively. These times are comparable to those obtained in preceding particle-interaction studies using the FMM on a GPU. The latest NVIDIA GPU architecture can achieve eight times the DP performance of the previous generation on which GTX295 is based. Thus, the code developed in this study could possibly run, unchanged, that much faster on the latest hardware.

Acknowledgements

This study was partially supported by the Japan Society for the Promotion of Science (JSPS). The author sincerely thanks T. Yamamoto, T. Sasayama, and T. Kobayashi for their contributions to this study.

Appendix A. Discretized boundary equations

On a boundary element that is not used as an electrode element, the following boundary equation is imposed: $\sigma_- F_- = \sigma_+ F_+$. The subscripts $\pm$ indicate that $\boldsymbol{E}$ and σ are defined on the plus or minus side of the element surface, respectively.

The elements of the electrodes are classified into four subsets: (i) a designated element of electrode 1, (ii) the other elements of electrode 1, (iii) a designated element of electrode 2, and (iv) the other elements of electrode 2. On these elements, the following boundary equations are imposed: $V_{(\mathrm{i})} = V_{(\mathrm{ii})}$; $V_{(\mathrm{iii})} = V_{(\mathrm{iv})}$; $\sum_{(\mathrm{i})(\mathrm{ii})} \sigma_{\mathrm{in}} F_{\mathrm{in}} = 1$; $\sum_{(\mathrm{i})(\mathrm{ii})(\mathrm{iii})(\mathrm{iv})} \sigma_{\mathrm{in}} F_{\mathrm{in}} = 0$. The symbols (i)–(iv) indicate that they are related to the elements of the subsets (i)–(iv), and F_{in} is defined as $\int \boldsymbol{E}_{\mathrm{in}} \cdot \boldsymbol{n}_{\mathrm{in}} dS$. The symbol "in" specifies the tissue side of the electrode element. The direction of $\boldsymbol{n}_{\mathrm{in}}$ is also defined as toward the tissue.

References

[1] http://www.nvidia.com/page/home.html
[2] N. A. Gumerov, R. Duraiswami, Journal of Computational Physics 227 (2008) 8290
[3] R. Yokota, T. Narumi, R. Sakamaki, S. Kameoka, S. Obi, K. Yasuoka, Computer Physics Communications 180 (2009) 2066
[4] L. Greengard, V. Rokhlin, Acta Numerica 6 (1997) 229
[5] N. A. Gumerov, R. Duraiswami, Fast Multipole Methods for the Helmholtz Equation in Three Dimensions (Elsevier, 2004)
[6] T. Takahashi, T. Hamada, International Journal for Numerical Methods in Engineering 80 (2009) 1295
[7] T. Takahashi, Journal of the Japan Society for Simulation Technology 28 (3) (2009) 125 (in Japanese)
[8] S. Hamada, T. Kobayashi, IEEJ Trans. FM 126 (5) (2006) 355 (in Japanese) (translation: Electrical Engineering in Japan, 165 (4) (2008) 1)
[9] S. Hamada, M. Kitano, T. Kobayashi, IEEJ Trans. FM 128 (4) (2008) 223 (in Japanese).
[10] R. F. Harrington, K. Pontoppidan, P. Abrahamsen, N. C. Albertsen, Proc. IEEE 116 (10) (1969) 1715
[11] S. Sato, W. S. Zaengl, Proc. IEEE 133 (2) (1986) 77
[12] R. F. Harrington, Proc. IEEE 55 (2) (1967) 136

[13] A. Hirata, K. Yamazaki, S. Hamada, Y. Kamimura, H. Tarao, K. Wake, Y. Suzuki, N. Hayashi, O. Fujiwara, Radiation Protection Dosimetry 138 (3) (2010) 237
[14] S. Hamada, GPU accelerated lead field calculation by indirect boundary element method for voxel models, Brain Topography and Multimodal Imaging (Proceedings of ISBET 2009), Kyoto University Press ISBN: 9784876987993 , (Oct. 2009) 187
[15] C. H. Choi, J. Ivanic, M. S. Gordon, K. Ruedenberg, Journal of Chemical Physics 11 (19) (1999) 8825
[16] T. Nagaoka, S. Watanabe, K. Sakurai, E. Kunieda, S. Watanabe, M. Taki, Y. Yamanaka, Physics in Medicine and Biology 49 (2004) 1
[17] S. Fujino, M. Fujiwara, M. Yoshida, Transactions of the Japan Society for Computational Engineering and Science 8 (2006) 145 (in Japanese)